软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(5):2196−2217 [doi: 10.13328/j.cnki.jos.006401]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

版本失配和数据泄露对基于缺陷报告的缺陷定位模型的影响*

周慧聪[1,2], 郭肇强[1,2], 梅元清[1,2], 李言辉[1,2], 陈 林[1,2], 周毓明[1,2]

[1](计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210093)

[2](南京大学 计算机科学与技术系, 江苏 南京 210023)

通信作者: 李言辉, E-mail: yanhuili@nju.edu.cn; 周毓明, E-mail: zhouyuming@nju.edu.cn

摘 要: 为了降低缺陷定位过程中的人力成本, 研究者们在缺陷报告的基础上提出了许多基于信息检索的缺陷定位模型, 包括使用传统特征和使用深度学习特征进行建模的定位模型. 在评价不同缺陷定位模型时设计的实验中, 现有研究大多忽视了缺陷报告所属的版本与目标源代码的版本之间存在的"版本失配"问题或/和在训练和测试模型时缺陷报告的时间顺序所引发的"数据泄露"问题. 致力于报告现有模型在更加真实的应用场景下的性能表现, 并分析版本失配和数据泄露问题对评估各模型真实性能产生的影响. 选取 6 个使用传统特征的定位模型 (BugLocator、BRTracer、BLUiR、AmaLgam、BLIA、Locus) 和 1 个使用深度学习特征的定位模型 (CodeBERT) 作为研究对象. 在 5 个不同实验设置下基于 8 个开源项目进行系统性的实证分析. 首先, CodeBERT 模型直接应用于缺陷定位效果并不理想, 其定位的准确率依赖于目标项目的版本数目和源代码规模. 其次, 版本匹配设置下使用传统特征的定位模型在平均准确率均值 (*MAP*)、平均序位倒数均值 (*MRR*) 两个指标上比版本失配实验设置下最高可以提高 47.2% 和 46.0%, CodeBERT 模型的效果也受到数据泄露和版本匹配的双重影响. 使用传统特征的缺陷定位模型的性能被低估, 而使用深度学习特征的 CodeBERT 模型在应用于缺陷定位任务时还需要更多的探索和验证.

关键词: 缺陷定位; 缺陷报告; 版本失配; 数据泄露; 信息检索

中图法分类号: TP311

中文引用格式: 周慧聪, 郭肇强, 梅元清, 李言辉, 陈林, 周毓明. 版本失配和数据泄露对基于缺陷报告的缺陷定位模型的影响. 软件学报, 2023, 34(5): 2196–2217. http://www.jos.org.cn/1000-9825/6401.htm

英文引用格式: Zhou HC, Guo ZQ, Mei YQ, Li YH, Chen L, Zhou YM. Watch out for Version Mismtaching and Data Leakage! A Case Study of Their Influence in Bug Report Based Bug Localization Models. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2196–2217 (in Chinese). http://www.jos.org.cn/1000-9825/6401.htm

Watch out for Version Mismtaching and Data Leakage! A Case Study of Their Influence in Bug Report Based Bug Localization Models

ZHOU Hui-Cong[1,2], GUO Zhao-Qiang[1,2], MEI Yuan-Qing[1,2], LI Yan-Hui[1,2], CHEN Lin[1,2], ZHOU Yu-Ming[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Abstract: In order to reduce the labor cost in the process of bug localization, researchers have proposed various automated information retrieval based bug localization models (IRBL), including those models leveraging traditional features and deep learning based features. When evaluating the effectiveness of IRBL models, most of the existing studies neglect the following problems: the software version mismatching between bug reports and the corresponding source code files in the testing data or/and the data leakage caused by the chronological order of bug reports when training and testing their models. This study aims to investigate the performance of existing

* 基金项目: 国家自然科学基金 (61772259, 61872177)
周慧聪和郭肇强为共同第一作者.
收稿时间: 2021-03-02; 修改时间: 2021-04-26; 采用时间: 2021-06-28; jos 在线出版时间: 2022-06-15
CNKI 网络首发时间: 2022-11-15

models in real experiment settings and analyzes the impact of version mismatching and data leakage on the real performance of each model. F irst, six traditional information retrieval-based models (Buglocator, BTRracer, BLUiR, AmaLgam, BLIA, and Locus) and one novel deep learning model (CodeBERT) are selected as the research objects. Then, an empirical analysis is conducted based on eight open-source projects under five different experimental settings. The experimental results demonstrate that the effectiveness of directly applying CodeBERT in bug localization is not as good as expected, since its accuracy depends on the version and source code size of a test project. Second, the results also show that, compared with the traditional version mismatching experimental setting, the traditional information retrieval-based models under the version matching setting can lead to an improviment that is up to 47.2% and 46.0% in terms of *MAP* and *MRR*. Meanwhile, the effectiveness of CodeBERT model is also affected by both data leakage and version mismatching. It means that the effectiveness of traditional information retrieval-based bug localization is underestimated while the application of deep learning based CodeBERT to bug localization still needs more exploration.

Key words: bug localization; bug report; version mismatching; data leakage; information retrieval

在软件的演化过程中, 软件缺陷的引入不可避免. 这些缺陷的存在会导致软件在工作时触发不可预料的崩溃, 进而引发违背设计预期的行为或者预期功能的丧失, 最终造成一些无法挽回的后果. 为了能及时降低软件缺陷带来的不利影响, 软件维护人员需要在软件产品的更新换代或版本更迭中不断对已经发现的缺陷进行定位和修复来提升软件的质量. 其中, 缺陷定位是进行软件维护的重要步骤, 后续的缺陷修复工作需要在定位到缺陷位置的前提下进行. 同时, 由于项目中的源代码体量通常十分巨大 (由成百上千的源文件组成), 手动进行定位需要消耗大量的人力成本[1]. 为了快速准确地定位缺陷, 研究者们提出了各种自动化的缺陷定位模型[2−6].

一种主流的缺陷定位技术即基于信息检索的缺陷定位 (information retrieval based bug localization, IRBL) 受到研究者的持续关注和研究[7−11]. 这类方法借助缺陷报告 (bug report) 提供的信息完成缺陷定位任务. 首先, IRBL 方法使用目标项目的源文件构建一个集成各种特征 (如版本历史和代码结构) 的检索系统. 当收到新的缺陷报告时, 它们从缺陷报告中提取关键的文本信息构建出一个查询语句, 然后基于文本相似性在检索系统中匹配可疑的源文件. 最后将可疑的源文件推荐给维护人员进行人工审查. 由于 IRBL 方法可以自动地进行可疑代码的推荐并且具有可接受的准确性, 同时, 这类方法的使用成本较低并且大多数开源[12], 因此 IRBL 方法一直受到研究者的追捧.

在缺陷定位中所依赖的缺陷报告大多都是由用户撰写并提交的, 因此相较于代码文本, 缺陷报告的内容更偏向于非结构化的自然语言. 近年来, 受到自然语言处理的启发以及深度学习发展的影响, 研究者开始使用深度学习方法去更进一步挖掘缺陷报告中的语义信息代替传统的特征来解决缺陷定位问题[13−17]. 相比于使用传统特征的 IRBL 模型, 使用深度特征的模型需要使用历史数据来训练定位模型进行预测. 根据现有研究的报告结果, 这些模型在某些项目 (例如 Zhou 等人[18]提供的数据集) 上表现出了不错的性能. 然而几乎所有深度特征的模型都是不开源的, 因此后来的研究者通常无法复现已有的模型.

根据对现有的缺陷定位文献中验证实验的观察, 现有实验在评估 IRBL 模型时大多存在以下问题, 可能会导致评估结果的失真.

(1) 版本失配问题. 版本失配是指实验中的缺陷报告和其对应的问题代码版本不一致[19]. 缺陷报告和代码库中的源文件都是与特定版本的软件相关的. 其中, 缺陷报告是由用户根据特定版本的软件在使用过程中引发的缺陷撰写并提交的. 在大多数情况下, 一个项目所对应的缺陷报告不属于同一个版本. 当需要通过缺陷报告来定位引起缺陷的源代码文件时, 有必要在缺陷报告对应的项目版本上构建检索系统进行可疑代码的查询. 项目源代码在演化过程中会更新换代多个版本, 版本之间的代码文件的内容差别也许很大, 而现有实验大多没有考虑缺陷报告和目标代码的版本匹配问题, 它们在将所有报告视为同一个版本的情况下进行缺陷定位, 这可能会导致评估实验结果的失真.

(2) 数据泄露问题. 数据泄露是指在实验中评估模型时用到了“未来”的信息[20,21]. 在应用有监督模型时需要大量的数据训练来进行模型的拟合以实现缺陷定位. 但是, 缺陷报告并不是一个简单的文本文件, 而是一种拥有时间属性信息的数据. 缺陷报告的创建时间由提交者决定. 在训练模型的时候, 如果使用“未来”的缺陷报告作为训练的数据来构建模型对“过去”的缺陷报告推荐可疑的源代码文件, 那么得出的结论很可能与实际情况不符. 在文献 [20,22] 中也表明, 数据泄露可能会造成实验结果盲目乐观. 现有实验在评价有监督模型的有效性时大多没有考

虑到缺陷报告所特有的时间属性. 这无疑会对实验的结果产生影响, 导致定位模型的评估结果出现偏颇.

上述两个问题都会给模型评估带来偏差, 然而在现有的模型评估中忽视了这两个问题[2−6,13−18], 或者只考虑了其中的一个问题[19,20]. 本文以传统的 IRBL 模型和基于深度学习的 IRBL 模型为例, 验证版本失配和数据泄露两个问题对于基于缺陷报告的缺陷定位模型的影响. 针对传统 IRBL 模型, 本文选取了 6 种模型 (BugLocator[18]、BRTracer[2]、BLUiR[4]、AmaLgam[5]、BLIA[6]和 Locus[3]), 这些模型在学术界广泛使用, 其代码已经被整理规范并且开源[19]. 针对有监督模型, 由于深度学习模型的不开源以及难以复现性, 我们无法使用现有的模型. 为了应用最新的技术, 在本文中, 我们使用新近提出的自然语言模型 CodeBERT[23]来构建有监督模型进行缺陷定位, 同时探寻使用基于 CodeBERT 的深度学习预处理模型在缺陷定位方面的可行性以及参数影响. 本文的主要贡献总结如下:

(1) 针对实验的设置, 整理了清洁的数据集 (https://github.com/abbycc/IRBL_dataset.git), 包含了 8 个开源项目、涉及 3 025 个缺陷报告、17 344 个源代码文件.

(2) 系统地分析对比了 6 种传统的 IRBL 模型和 CodeBERT 模型在不同实验设置下的缺陷定位效果, 分析版本失配与数据泄露两个因素对各模型定位效果的影响.

(3) 本文总结了 CodeBERT 模型所存在的限制, 针对模型改进提供了建议.

本文第 1 节介绍缺陷定位的研究背景以及相关的研究工作. 第 2 节阐述现有研究在进行实验评估中存在的问题. 第 3 节给出本文的实验设计, 包括实验数据集、实验设置、研究问题以及评估指标. 第 4 节报告详细的实验结果并回答每个研究问题. 第 5 节分析该实证研究中存在的效度威胁. 第 6 节对本文的实验进行总结并对未来的研究点进行展望.

1 研究背景

本节首先介绍基于信息检索的缺陷定位的基本知识. 然后分别介绍本文研究的 6 种使用传统特征的定位模型和最新的基于深度学习框架 CodeBERT 的定位模型.

1.1 基于信息检索的缺陷定位

信息检索技术被广泛用来构建完成查询任务的模型. 这类模型的工作思路是计算查询语句与目标文档之间的匹配分数来推荐待求的结果. 在缺陷定位领域中, 利用信息检索技术去寻找产生缺陷的代码实体是目前主流的静态缺陷定位模型. 在这类模型中, 缺陷报告被视为查询语句, 源代码文件组成了目标文档库, 其定位过程可以描述如下.

(1) 构建语料库: 首先将每个源代码文件内容进行预处理. 对于某些程序关键词 (如 int, double) 和无意义的停用词 (如 a 和 the 等) 进行移除, 再通过词干分析法将变量名进行单词分割并将单词缩减为词根形式 (如 isCommitable 还原为 is 和 commit). 所有经过处理的词语构成了语料库.

(2) 构建索引: 对所有的源代码文件建立索引. 索引可以帮助模型定位到指定的文件, 并根据文件与查询语句的相似度进行排序.

(3) 构建查询: 在缺陷定位中, 使用缺陷报告来构建查询语句. 通过从缺陷报告中抽取所需要的信息 (如标题、描述等), 再使用构建语料库时用到的预处理方法生成最后的查询语句.

(4) 检索和排序: 在基于信息检索缺陷定位模型中, 查询语句会与文档库中的每一个源代码文件都生成一个分数来衡量该文件与查询语句的相关程度. 根据相关程度的得分对所有的文档进行降序排序, 最后反馈给软件维护人员.

维护人员根据模型反馈的排序列表对每个源文件逐个进行审查, 寻找与缺陷相关的文件. 可以看出, IRBL 模型如果可以将包含缺陷的源代码文件尽可能地排列在靠前的位置, 维护人员审查代码所需花费的工作量会减少, 整个缺陷定位及修复的进程将被加速.

1.2 使用传统特征的模型

传统特征指的是研究者通过观察缺陷实例在模型中加入的人工设计的特征, 主要包括版本历史、相似报告、

代码结构、堆栈踪迹和文本分段相关的特征. 根据我们前期工作的汇总[7], 以下 6 个使用传统特征的定位模型 (见表 1) 在学术界受到广泛地关注和研究.

表 1 调研的 IR-based 缺陷定位模型汇总

方法名称		BugLocator	BRTracer	BLIA	BLUiR	AmaLgam	Locus
年份		2012	2014	2017	2013	2017	2016
IR模型		rVSM	rVSM	rVSM	Indri	Mixed	VSM
模型特征	版本历史	—	—	√	—	√	√
	相似报告	√	√	√	√	√	—
	代码结构	—	—	√	√	√	√
	堆栈踪迹	—	√	√	—	—	—
	文本分段	—	√	√	—	—	—

• BugLocator. Zhou 等人[18]在他们的模型 BugLocator 用到了已修复的相似缺陷报告, 同时根据改进的空间向量模型 (rVSM) 计算缺陷报告与源代码文件的相似度. 对于新收到的缺陷报告, 首先计算该报告与历史上已经解决的缺陷报告之间的相似度; 再使用 rVSM 模型计算出该缺陷报告与每个源文件的文本相似度分值; 接着, 对历史上有缺陷的文件根据其对应的历史缺陷报告进行加权, 得到这部分文件的历史特征分数; 最后结合两部分的分数, 作为该缺陷报告与源代码文件的相似得分.

• BRTracer. Wong 等人[2]发现过大的源文件会降低 VSM 模型的准确性. 因此, 他们提出了 BRTracer 将每个的源文件按照某一阈值 (例如 100 行代码) 分为若干规模相同的片段, 并在此基础上计算每个片段与缺陷报告的相似性. 然后, 用得分最高的片段代表该文件. 与此同时, BRTracer 模型会匹配缺陷报告中堆栈踪迹里出现的文件名等代码实体来提升缺陷定位的准确性.

• BLUiR. Saha 等人[4]在 BLUiR 模型中利用从缺陷报告中提取的代码实体, 如类、方法和变量名来提高定位准确性. 缺陷报告有许多字段 (描述和摘要), 源代码文件也由许多部分组成 (类名、方法名、变量名和注释). BLUiR 将缺陷报告和源代码文件转换为结构化文档, 并使用信息检索模型 Indri[24]进行结构化信息检索. 在结构化信息检索中, 缺陷报告中每个字段的文本内容和源代码文件的每个部分都是分开考虑的.

• AmaLgam. Wang 等人[5]在 AmLgam 模型中除了分析相似的报告和提取代码实体外, 还利用历史版本信息来帮助缺陷定位. AmaLgam 模型只考虑最近的版本历史, 并完全丢弃距离提交新缺陷报告超过 k 天的历史信息. 版本历史组件利用版本历史信息对文件进行排序, 再根据计算得出的相似的报告和代码结构的得分, 为每个源代码文件输出一个相似度分数.

• BLIA. Youm 等人[6]在他们的模型 BLIA 中集成了 Wong 等人[2]对堆栈踪迹的处理方法. 同时, BLIA 整合了之前提出的与缺陷相关的特征, 将相似的缺陷报告、修订历史、代码实体和堆栈踪迹信息组合在一起, 以提高 BLIA 模型的准确性.

• Locus. Wen 等人[3]提出了 Locus 模型. 他们发现孤立地定位有缺陷的模块可能不能为开发人员提供足够的信息来理解和修复缺陷, 而缺陷引入变更 (bug-inducing change) 信息有可能解决这个问题. Locus 对自然语言和代码实体分别构建语料库和查询语句, 将两个查询结果组合起来输出变更块 (change hunk) 的排序, 并在此基础上选出文件或者变更, 该模型在目前获得较好的性能.

这 6 种使用传统特征的模型被用于本文的实证研究中, 每种模型的默认参数与原文保持一致. 虽然对于 IRBL 的相关研究[3,25]表明, 每种 IRBL 模型的配置和参数可能会对缺陷定位的性能产生重大影响. 但本文的目标是复现相关的模型和结果, 因此对每一种模型使用相同的配置和参数, 保证实验的客观性.

1.3 使用深度特征的模型

随着深度学习技术的发展, 近年来研究者们将深度学习技术引入基于信息检索的缺陷定位领域[13−15,23]. 相较于传统模型中用到的人工设计的传统特征, 这类模型是通过训练深度学习网络 (如 CNN、RNN 等) 捕捉缺陷报告

和源代码文件之间潜在的联系, 提取出文本中更深层、更高维的抽象特征. 这种使用浅层或深层神经网络对查询所反馈的搜索结果进行排序的模型称为神经信息检索模型 (neural information retrieval-based model)[26]. 下面是近年提出的相关模型.

• DNNLOC. Lam 等人[13]将深度神经网络 (DNN) 与信息检索技术中的 rVSM 模型相结合进行缺陷定位. rVSM 部分计算缺陷报告和源代码文件之间文本相似性. DNN 部分学习缺陷报告中的术语与源代码中的代码字段之间的联系.

• BugTranslator. Xiao 等人[14]提出一个类似机器翻译的深度学习方法用于缺陷定位. 该方法由基于注意力的循环神经网络 (RNN) 编码器和解码器组成. 一个 RNN 将缺陷报告编码为几个向量, 再由另一个 RNN 解码为缺陷报告所对应的代码形式, 以此来确定与缺陷报告相关的源代码文件. 该技术研究并采用从缺陷报告中提取的语义信息与源文件之间的相关性信息.

• CNN_Forest. Xiao 等人[15]使用卷积神经网络和级联森林对缺陷报告和源代码文件的语义信息和结构特征建模. 通过具有多个过滤器的卷积神经网络和具有多粒度扫描的随机森林集合, 分别从缺陷报告和源文件中提取词向量的语义和结构特征, 随后从级联森林进一步提取更深层次的关联特征用于缺陷定位.

以上模型都基于神经网络进行建模. 虽然在他们的原始论文中报告了较好的性能, 我们发现这些模型在实际应用时具有以下 3 点挑战.

(1) 基于神经网络的模型在搭建时需要设置大量的没有详细说明的参数, 因此这类模型的构建比较复杂.

(2) 几乎所有原始文章中都没有完全将其代码开源, 因此这类模型很难进行复现.

(3) 这类模型的训练具有一定的随机性, 即便是完全相同的设置下, 每次的训练结果也不完全一致.

因为存在这些挑战, 进行模型比较时, 之前的研究者所复现出的基于深度学习的模型在缺陷定位性能上与原始文中的报告的数据有差异[15]. 为了避免复现误差造成的不可靠结论, 我们在本文中不再使用这些模型.

1.4 CodeBERT 缺陷定位模型

Feng 等人[23]提出了一个最新的开源预训练模型 CodeBERT 用于自然语言代码搜索和代码文档生成等领域的建模并取得了不错的结果. CodeBERT 是第一个针对多种编程语言的大型 NL (自然语言)-PL (编程语言) 的预训练模型. 它可以捕获自然语言和编程语言之间的语义联系, 并生成通用表示, 这些表示可以广泛支持 NL-PL 理解任务 (如自然语言代码搜索) 和生成任务 (如代码文档生成). 结合利用缺陷报告进行缺陷定位的应用场景, 本文基于 CodeBERT 构建了一个缺陷定位模型, 并将其与传统模型进行比较来探寻该模型在缺陷定位方面的可行性.

1.4.1 CodeBERT 模型架构

CodeBERT 模型遵循 BERT[27]和 RoBERTa[28]的架构, 并使用多层双向的 Transformer[29]作为 CodeBERT 的模型组件. 本文中不再详细回顾无处不在的 Transformer 架构, 而是从模型输入和训练任务两个方面简要介绍 CodeBERT 模型的架构.

(1) 输入. CodeBERT 的输入由自然语言文本、特定的编码语言代码和特殊分隔符组成. 它将自然语言文本和代码语言文本均视作一个词序列. 假设自然语言序列中第 n 个单词为 w_n, 编码语言代码中第 m 个单词为 c_m, 则输入的格式为: <[CLS], w_1, w_2,…, w_n, [SEP], c_1, c_2,…, c_m, [EOS]>. 其中, [CLS] 是输入首部的一个特殊的令牌 (token), 该令牌的值代表自然语言文本与代码语言文本之间的语义相关性. 该值可以是区间 [0, 1] 之间的数用于完成排序任务, 也可以通过 Softmax 函数转换为集合{0, 1}中的某个数用于完成分类任务 (其最终的数值根据具体的任务而定). 令牌 [SEP] 和 [EOS] 分别标记自然语言文本序列和代码单词序列的结束.

(2) 训练任务. 为了学习文本的语义信息与自然语言和编程语言之间的联系, CodeBERT 模型在训练时要实现以下两个训练目标: 屏蔽语言建模 (masked language modeling, MLM) 和替代标记检测 (replaced token detection, RTD).

① 屏蔽语言建模 (MLM). 该任务的训练目的是通过未屏蔽的词去预测被屏蔽的词. 首先将 NL-PL 对构成数据对 (x={w, c}) 作为模型的输入. 其中, w 是自然语言 (NL) 的序列, c 是程序语言 (PL) 的序列. 屏蔽语言是指选择

NL 序列和 PL 序列中的一个随机的位置 (如 m^w 和 m^c, m 是指被屏蔽的位置), 将该位置的单词替换为一个特殊的 (MASK) 令牌. 根据 Devlin 等人[27]推荐的参数, 需要对 x 中屏蔽 15% 字节的内容. 根据如下公式计算模型的损失函数, 其中, p^{D_1} 表示从大量词汇表 D_1 中预测被屏蔽词的概率; w^{masked} 和 c^{masked} 分别表示被屏蔽的单词, θ 为模型中的相关参数.

$$\ell_{\text{MLM}}(\theta)=\sum_{i\in m^w\cup m^c}-\log p^{D_1}(x_i|w^{\text{masked}},c^{\text{masked}}) \tag{1.1}$$

② 替代标记检测 (RTD). 该任务是生成替换词来判别是否是原词. 为此, 训练数据为单峰数据 (不成对的 NL 或 PL). 由于 RTD[30]最初是为了有效地学习预先训练好的自然语言模型而开发的, 因此 CodeBERT 采用 RTD 建模, 能够达到可以同时使用双峰和单峰数据进行训练的目的. 替代标记检测 (RTD) 模型中包含两个数据生成器: NL 生成器 p^{Gw} 和 PL 生成器 p^{Gc}. 在该目标中, 两个生成器分别为自然语言和代码语言的随机位置生成替换的单词, 然后再以二分类的方式训练识别器来判断替换的单词是否是原词.

图 1 为替代标记检测 (RTD) 的一个实例. 在图中, w 序列代表自然语言 (NL), c 序列代表程序语言 (PL). 自然语言中第 1、5 位, 程序语言中第 2、6 位分别被 NL 生成器和 PL 生成器替换为其他单词 (有一定的概率生成原单词). 例如, w_1 被替换为新单词 w_{51}, 而 w_5 仍然被替换为 w_5. 经过替换后的新序列 w 和 c 将会再次输入 NL-Code 生成器, 该生成器会识别新序列中的单词是否为原词.

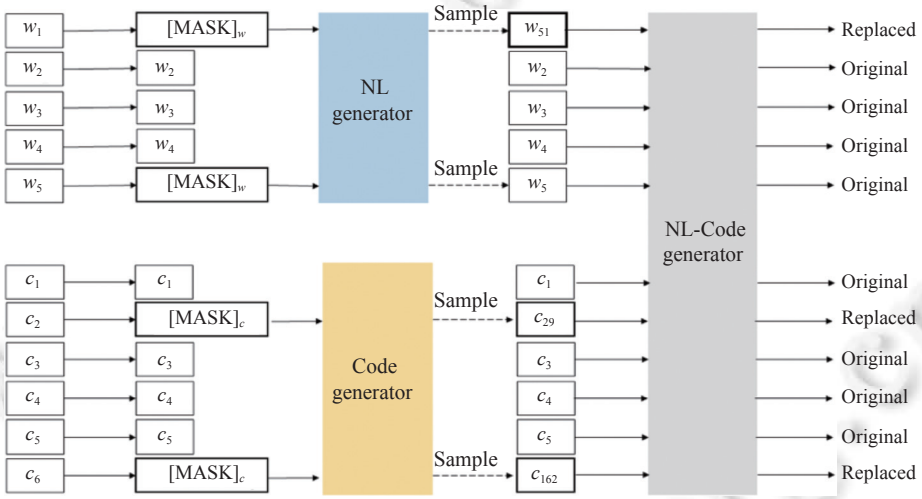


图 1 替代标记检测的一个实例

通过在大量代码和自然语言文本上训练屏蔽语言建模 (MLM) 和替代标记检测 (RTD) 这两个任务, CodeBERT 模型可以更好地理解输入文本的语义信息, 也可以捕捉自然语言和代码语言的联系. 因此, 在该已训练好的预训练模型的基础上, 只需要针对下游任务对预训练模型进行微调, 便可以将 CodeBERT 应用到各类检索任务中.

1.4.2 CodeBERT 定位模型

图 2 描述了应用 CodeBERT 进行缺陷定位的过程, 该过程包含训练和测试两个阶段. 在训练阶段, 首先将缺陷报告看作是自然语言文本进行预处理 (分词、删除停止词). 然后, 将预处理过后的报告文本与源代码文件相连接, 形成一个句子对实例, 根据该缺陷报告与相连接的源代码文件是否相关对该实例进行标记. 如果该报告与该源代码文件相关, 则标记为 1, 否则标记为 0. 这样就生成了一个标记好的输入三元组< 缺陷报告-源代码文件-标签>. 最后, 将该三元组输入 CodeBERT 模型进行训练.

在测试阶段, 首先, 与训练阶段类似, 将待定位缺陷报告与目标项目中的每一个源代码文件都连接起来, 形成一个二元组句子对<测试报告-源代码文件>输入模型. 然后, 利用 CodeBERT 模型对句子对进行计算, 并输出一个结果向量 $[p_b, p_c]$, 其中的元素分别表示该句子对被预测为 1 和 0 的概率. 若该句子对标记为 1 的概率越大, 说明缺

陷报告与该源代码文件相关性越高. 最后, 根据每个句子对被预测为 1 的概率对所有句子对进行降序排序, 这样就可以获得与该缺陷报告相关的源代码文件可疑度推荐列表.

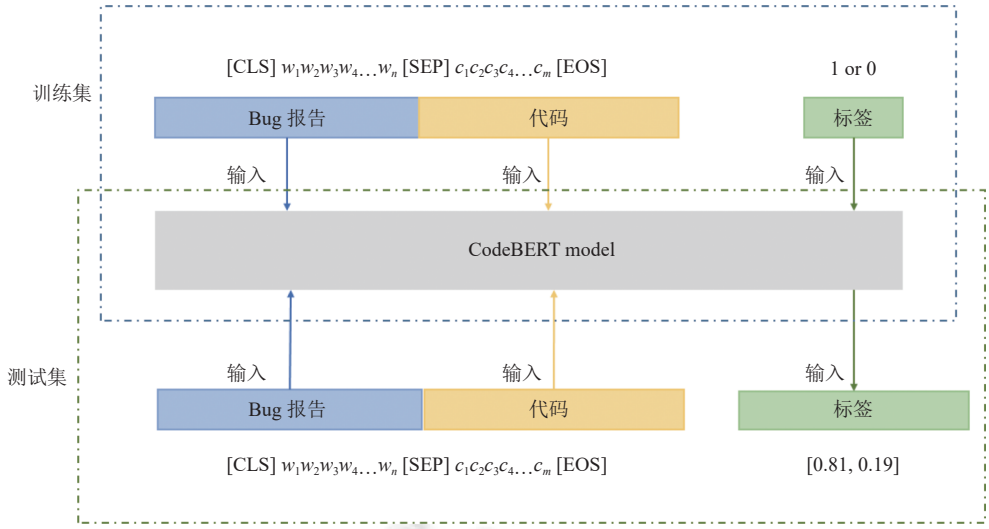


图 2 使用 CodeBERT 进行缺陷定位的流程

2 传统实验设置的问题

目前学术界在对基于信息检索的缺陷定位模型进行实验评估的时候, 在实验设置上大多存在两个问题: 其一是版本失配问题, 其二是数据泄露问题. 本节首先介绍传统的实验设置, 然后依次探讨存在的两个问题.

2.1 传统实验设置

根据缺陷定位模型是否需要训练集, 可以将目前的缺陷定位模型分为无监督模型和有监督模型两类. 在对无监督的定位模型进行评估时, 现有文献[2−6,18]中一般采用将目标项目中所有的缺陷报告在同一版本的源代码文件上进行缺陷定位实验. 根据所有生成的推荐列表进行评估指标计算和对比, 以此来衡量模型的优劣.

在对有监督的定位模型进行评估时, 现有文献[13−17]通常采用简单的交叉验证的方法进行训练、验证和测试. 他们将所有的缺陷报告按比例分割为训练集和测试集. 对于训练集中的缺陷报告, 将每一个报告与项目中的每一个源代码文件关联起来, 并标记是否相关, 传入模型进行训练. 对于用于测试的缺陷报告, 根据预测模型计算出其与项目中的每一个源代码文件相关的概率. 无论是进行模型的训练还是测试, 所用于缺陷定位的源代码文件都属于同一个版本.

2.2 问题 1. 版本失配

软件项目在更迭过程中会产生许多不同的版本, 不同版本之间的区别很大. 例如, 随着版本的更新, 项目的功能不断地扩充和完善, 其对应的源代码文件数量会越来越庞大 (图 3 统计了 ROO 项目在 17 个版本上的文件数目). 现有的实验设置大多采用在单个版本上进行缺陷定位 (如图 4), 很容易出现缺陷报告和其对应的问题代码库版本不一致[19]的问题. 在该实验设置下, 缺陷定位将在单一的代码版本上完成. 因为在同一个版本上定位, 不需要单独将缺陷报告和所对应版本的源代码进行匹配, 其搜索空间比较简单并且工作量较少, 所以被广泛使用. 本文中涉及的 6 种采用传统特征的缺陷定位模型也在提出的论文中使用了该实验设置[2−6,18].

然而, 缺陷报告和代码库中的源文件都是与特定版本的软件相关的. 其中, 缺陷报告是由用户根据特定版本的软件在使用过程中引发的缺陷撰写并提交的. 用户在撰写缺陷报告时, 缺陷报告所对应的版本是用户正在使用的版本. 在大多数情况下, 缺陷报告的目标版本是提交时最新发布的版本[19]. 因此, 一个项目所对应的缺陷报告不属

于同一个版本. 当需要通过缺陷报告来定位引起缺陷的源代码文件时, 有必要在与缺陷报告对应的项目版本上构建检索系统进行可疑代码的查询. 将所有报告视为同一个版本的情况下进行缺陷定位, 这可能会导致评估实验结果的失真. 例如, 有可能版本 1.0 时提交的缺陷报告相关的源代码文件在版本 2.0 中已被修复或删除; 也有可能版本 2.0 时提交的缺陷报告所描述的缺陷在版本 1.0 的源代码文件中不存在. 在此情况下, 无论是在版本 1.0 的源代码文件上进行缺陷定位还是在版本 2.0 的源代码文件上进行缺陷定位, 都可能会导致缺陷定位模型根本无法定位到目标源代码文件, 从而会降低缺陷定位模型的性能. 同时, 不含有 bug 的源代码文件 (干净文件) 在版本的更迭中也会发生较大的变化, 而在缺陷定位过程中, 需要缺陷报告与大量的干净文件进行比较, 以此生成推荐列表. 干净文件的变化也会导致最后定位结果的变化. 由此可知, 版本失配所引起的问题可能会导致不准确的缺陷定位结果, 从而影响缺陷定位模型有效性的评估.

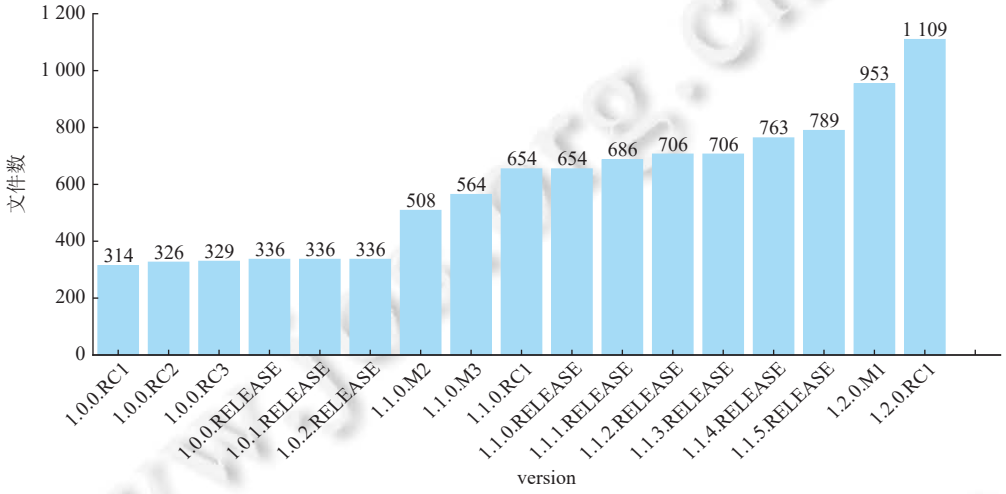


图 3 ROO 项目中不同版本源代码的文件数量

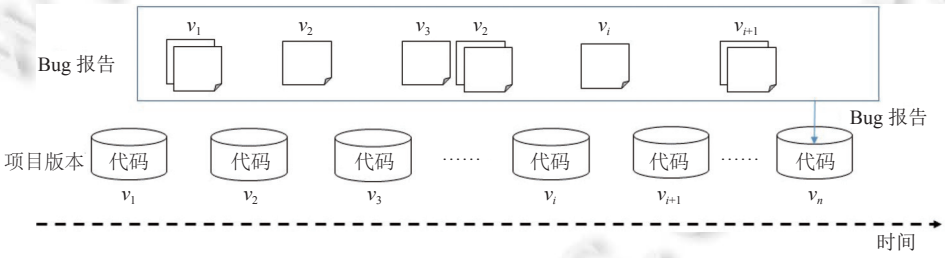


图 4 现有研究的实验设置中的版本匹配策略

2.3 问题 2. 数据泄露

数据泄露是缺陷定位中另一个十分重要的问题[20,22,31−34]. 本文中的数据泄漏指在实验中评估模型时用到了“未来”的信息, 而从“未来”泄漏的信息可能会导致预测模型产生误导性的乐观[20]. 该数据泄漏问题是由于忽略了产生数据的时间顺 序造成的. 同时, 它也是缺陷报告与源代码文件版本失配问题的一个延伸. 版本失配是时间顺序紊乱导致数据泄露的一个表现, 但在版本匹配的情况下, 依旧会存在数据泄露的问题. 软件的版本更迭与缺陷报告上传到系统是两条并行的时间线, 因此版本较前的软件所对应的缺陷报告的提交时间不一定早于版本较后的软件所对应的缺陷报告, 反之亦然.

图 5 为项目 MATH 中某些缺陷报告的时间序列, 由图 5 可知, 一个软件项目接收到的缺陷报告的时间顺序与其所所对应的软件版本的发布时间顺序没有联系. 例如, 版本 1.2 的发布时间早于版本 2.0, 然而与版本 1.2 有关联

的 ID 为 294 缺陷报告的提交时间 (2009.09.11) 却晚于与版本 2.0 有关联的 ID 为 272 的缺陷报告 (2009.05.30). 因此, 如果简单地使用与训练版本 (如 1.1) 关联的所有缺陷报告构造预测模型, 并使用该模型在测试版本 (如 2.0) 的源代码文件上进行缺陷定位, 那么该实验过程实际上已经产生了数据泄露, 即用到了在“未来时间”才能获取的数据. 例如, 当 2009 年 1 月 16 日收到软件版本 2.0 对应 ID 为 240 和 238 的缺陷报告时, 如果版本 1.1 对应的所有缺陷报告参与到建模过程中, 那么在 2009 年 10 月 24 日才能收集到的 ID 为 306 的缺陷报告则属于泄露的数据. 由于在 2009 年 1 月 16 日时系统还未收到 ID 为 306 的缺陷报告, 在实际的预测过程中是无法将其用于构建预测模型的. 之前的研究者[2,4,7]在实验时由于已经获取了完整的实验数据集, 从而可以在理论上构建一个不考虑数据泄露的预测实验. 但是在这种实验场景下评估得到的性能不能够完全反映一个定位模型在真实应用场景下的性能.

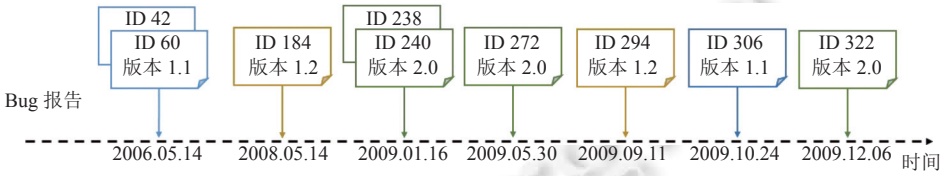


图 5 MATH 项目中不同版本缺陷报告的时间序列样例

在对于需要数据训练的缺陷定位模型, 如基于深度学习的缺陷定位模型, 如果训练数据的时间迟于测试数据, 显然是逻辑错误的. 如果使用这些“未来”的缺陷报告作为训练的数据来构建模型对“过去”的缺陷报告推荐可疑源文件, 那么得出的结论很可能与实际情况不符. 例如, Pendlebury 等人[22]对 Android 中间件分类器的实证研究结果表明, 在不考虑时间顺序进行训练集与测试集数据的划分时会 (人为) 放大分类器的分类性能, 从而得出有偏的结论. 相似地, 现有实验在验证使用深度学习特征的缺陷定位模型时没有考虑到缺陷报告所特有的时间属性. 这无疑会对实验结果的准确性产生影响. 本文的另一个目标是对这种影响进行验证和评估.

3 实验设计

本节介绍该实证的实验设计, 包括数据集、实验设置、评估指标和研究问题 4 部分内容.

3.1 实验数据集

3.1.1 筛选与划分

在 2018 年, Lee 等人[19]在他们的研究中收集了一个大型的缺陷定位数据集 Bench4BL. 该数据集中共包含了 46 个新的 Java 开源项目 (61 431 个 Java 源文件) 以及在这些项目上所收集到的 9 459 个缺陷报告. 由于该数据集有较大的规模、涵盖了较多的开源软件, 并且该数据集已被多项研究工作所采用[35−37], 因此本文实验中所使用的项目均取自 Bench4BL 数据集.

由于本文的实验需要模拟真实的缺陷定位的环境, 某些项目情况比较特殊 (如包含的缺陷报告数目太少), 无法模拟出真实的情况. 因此, 我们需要对该数据集中的项目进行筛选和整理的预处理操作, 从而选择合适的项目进行实验. 以下是我们的预处理步骤.

(1) 过滤掉缺陷报告数目小于 100 的项目. 报告数目过少会造成评估结果不客观 (偏高或者偏低). 例如, 考虑极端的情况, 若某项目仅包含一个缺陷报告, 那么一个定位模型在该项目上的召回率就是 1 或者 0, 均不能准确体现模型的真实性能.

(2) 根据缺陷报告中记录的版本信息对于缺陷报告按版本进行划分, 统计同属一个版本的缺陷报告数量和版本的数量, 并过滤掉在源代码仓库中不对应的版本的缺陷报告.

(3) 对于每一个项目, 去除掉单个版本中报告数目少于 10 的项目版本. 与 (1) 同理, 如果该版本中对应的报告数目太少, 该版本上的结果将不够准确, 无法反映真实的定位情况. 同时, 某些项目的版本太多, 使用基于深度学习的方法进行评估时需要耗费大量的工作量, 因此本文中只选取版本数目少于 25 的项目进行测试.

(4) 根据第 3.2 节中的实验设置和每个版本的提交时间划分训练集和测试集, 过滤无法提取相应训练集和测

试集的版本.

在得到合适的测试的项目和需要进行缺陷定位的版本后, 根据以下步骤实现在同一版本下进行缺陷定位.

① 使用 git 工具获取项目中所有的 tag 信息, 获取每个项目版本名以及对应的最新提交时间. 这些时间也是在之后将缺陷报告划分训练集和测试集的基准.

② 将从 tag 中提取的版本信息与缺陷报告中提取的版本信息进行对照, 过滤在 tag 中找不到对应版本的缺陷报告, 确保每个缺陷报告都可以找到与它的版本对应的源代码文件.

在筛选出的项目集上针对传统的 IRBL 方法和基于深度学习的 CodeBERT 模型设计相关的实验设置, 具体实验设置描述见第 3.2 节. 对于传统的 IRBL 方法, 设计两种版本失配和版本匹配的实验对照组来验证版本失配对于该缺陷定位模型的影响. 对于 CodeBERT 模型设计 3 种实验设置来分析版本失配和数据泄露两个因素对于定位模型的影响.

3.1.2 数据详情

表 2 总结了经过筛选过后本文中使用的实验项目. 这些项目来自于 Apache2、Spring3 和 JBoss4 的软件社区. 与以往研究设置一致[4,7−9], 本文只从每个项目收集源代码文件 (*.java 文件). 根据第 3.1.1 节的步骤过滤后剩下 8 个测试项目, 其中包含 3 025 个缺陷报告、17 344 个源代码文件和 120 个项目版本. 由于在同一个项目中版本不同, 所对应的源代码文件数量也不同. 源代码文件的数量是根据每个项目的源文件的最大数量进行统计的.

表 2 本文中选取的项目

项目名称	缺陷报告数目 (过滤后)	源代码文件数量 (最大)	版本数目 (过滤后)
HBase	477	2 714	18
HIVE	852	4 651	23
LANG	181	305	9
MATH	217	1 617	9
WFCORE	111	3 598	9
BATCH	279	1 732	14
ROO	615	1 109	19
SEC	293	1 618	19
总计	3 025	17 344	120

注: 与文献[19]中报告的不同，他们在网上共享的数据集中，HIVE项目实际包含23个版本，ROO项目实际包含21个版本

3.2 实验设置

本节介绍各研究模型的实验设置. 由于本文中要评估的 7 个基于信息检索缺陷定位模型, 包括 6 个无监督的使用传统特征的模型和一个有监督的使用深度学习特征的模型 (CodeBERT). 因此本文的实验根据模型的特性分为两大类 5 种具体实验设置.

3.2.1 无监督模型的实验设置

使用传统特征的无监督模型不需要进行模型的训练, 因此只需要在分版本的情况下对项目进行缺陷定位, 便可以获得在真实情况下 (无版本失配问题) 各个模型的真实定位性能. 为了同之前研究的实验结果进行对比, 无监督模型在以下两个设置下进行实验.

(1) 版本失配 (对照组): 不考虑缺陷报告和源代码文件所包含的版本信息, 所有版本的报告在同一个版本的源代码上进行缺陷定位实验.

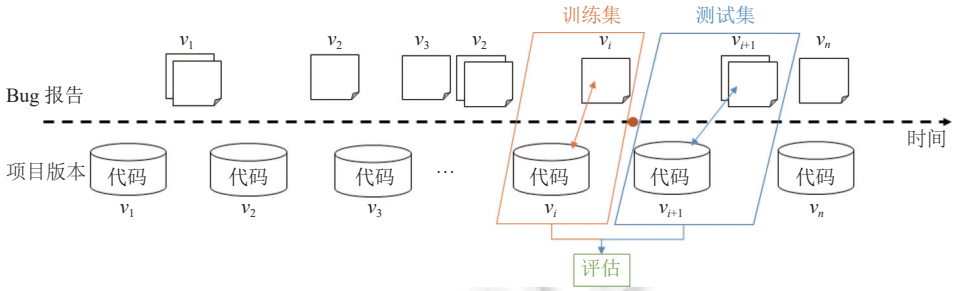
(2) 版本匹配 (实验组): 根据缺陷报告的版本信息, 在对应版本的源代码上进行缺陷定位, 确保源代码和缺陷报告的版本一致.

3.2.2 CodeBERT 模型的实验设置

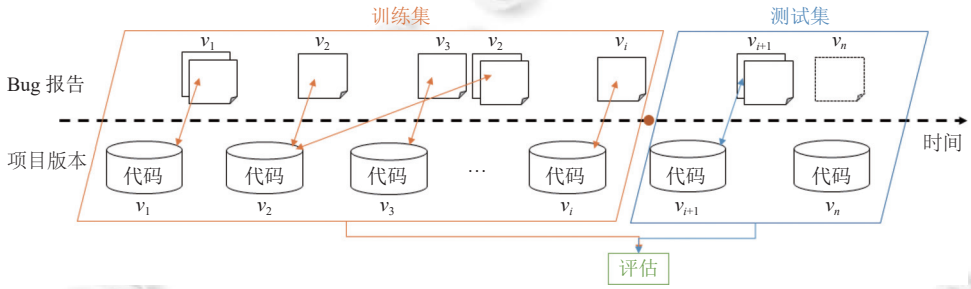
基于深度学习的定位模型性能依赖于训练数据, 因此根据第 3.1.1 节中使用 git 工具所获取的每个项目版本名以及对应的最新提交时间, 将所有的项目版本根据最新提交时间进行递增排序 (时间早的版本排在前面), 确保用

于训练的缺陷报告提交时间早于用于测试的缺陷报告. 基于深度学习的方法 (CodeBERT) 的评估实验采用以下 3 种设置划分训练集和测试集.

(1) 邻近版本的实验设置 (图 6(a)): 对某个项目的待测版本 v_{i+1}, 将 v_{i+1} 的提交时间作为一个划分节点 δ. 按该时间节点对前一个版本 v_i 的缺陷报告进行筛查, 将早于时间节点 δ 的缺陷报告划为训练集合 (缺陷报告中的版本字段值靠前不一定提交时间早). 再用该时间节点 δ 对当前版本 v_{i+1} 的缺陷报告进行筛查, 将晚于时间节点 δ 的缺陷报告划为测试集 (缺陷报告不一定全部晚于版本最新的提交时间).



(a) 邻近版本的实验设置



(b) 全部版本的实验设置

图 6 新增的 CodeBERT 定位模型实验设置

(2) 全部版本的实验设置 (图 6(b)): 对某个项目的待测版本 v_{i+1}, 将 v_{i+1} 的提交时间作为一个划分节点 δ. 按该时间节点对 v_{i+1} 之前的所有版本 (v_1, v_2, …, v_i) 的缺陷报告进行筛选, 早于该时间节点的缺陷报告全部选为训练集. 再对剩下的所有版本 (v_{i+1}, v_{i+2}, …, v_n) 的缺陷报告进行筛选, 晚于该时间节点的缺陷报告选为测试集.

(3) 传统的实验设置 (对照组): 按照简单的划分测试, 不考虑缺陷报告的时间顺序, 将缺陷报告随机进行划分, 将 90% 的缺陷报告用于训练, 10% 的缺陷报告用于测试, 实验重复进行 10 次.

在确定了训练集和测试集后, 将每一个缺陷报告的版本信息进行记录. 当使用缺陷报告进行模型训练和测试时, 使用 git 工具将项目源代码回滚到与缺陷定位一致的版本上. 在缺陷报告与源代码文件版本相同的情况下, 对模型进行训练和测试.

为了公平地比较传统的 IRBL 模型和 CodeBERT 模型, Code BERT 实验中的第 (1)、(2) 种设置被设定为版本匹配的情况 (即缺陷报告与源代码文件版本一致); 第 (3) 种设置被设定为版本失配情况 (即所有的测试都在同一版本下进行, 缺陷报告与源代码文件版本不一定相同). 在这种设置下, 将传统的 IRBL 定位模型与 CodeBERT 定位模型进行对比.

3.3 评估指标

当评估 IRBL 技术时, 普遍采取将实验结果与 ground truth (真实值) 比较进行评估. ground truth 数据包括开发人员已经修复的 bug 报告和相关的源代码文件 (为修复 bug 而修改的文件). 在数据集上应用 IRBL 技术, 并将输出结果与该数据集的 ground truth 进行比较. 在以往的研究中[2−6,18], 以下评估方法性能的指标被广泛应用.

(1) 平均准确率均值 *MAP* (mean average precision). 该指标测量的是缺陷定位模型定位到的所有与缺陷报告相关的源代码文件在排序列表中的位置.

$$\begin{cases} MAP = \dfrac{1}{n}\sum_{j=1}^{n} AvgP_j \\ AvgP_j = \dfrac{1}{|K_j|}\sum_{k\in K_j}\{Prec@k\} \\ Prec(k) = \dfrac{\sum_{i=1}^{k} IsRelevant(k)}{k} \end{cases},$$

其中, n 为缺陷报告的总数; 对于第 j 个缺陷报告, $Prec@k$ 表示所对应推荐列表的前 k 个文件里与缺陷报告相关的源代码文件所占的比例; $IsRelevant(k)$ 表示推荐列表中第 k 个源代码文件是否与缺陷报告相关, 相关为 1, 不相关为 0.

(2) 首位倒排均值 *MRR* (mean reciprocal rank). 该指标测量的是缺陷定位模型定位到的首个与缺陷报告相关的源代码文件在排序列表中的位置.

$$MRR = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{rank_j},$$

其中, $rank_j$ 表示第 j 个缺陷报告所对应的推荐列表中首个与缺陷报告相关的源代码文件的排名.

3.4 研究问题

为了研究 IRBL 模型在真实缺陷定位的情景下所表现出来缺陷定位性能, 本文设计了以下 3 个研究问题.

(1) RQ1: 与传统的 IRBL 模型相比, CodeBERT 缺陷定位的效果如何?

该研究问题的目的是验证 CodeBERT 模型在缺陷定位方面的效果. 目前基于信息检索的缺陷定位模型较为成熟和主流, 与通过与使用传统特征的 IRBL 模型对比, 可以直观地体现出 CodeBERT 模型被应用于缺陷定位任务时表现出的性能. 该研究问题的对比实验分为 3 部分: ① 版本失配设置下定位效果的对比; ② 版本匹配设置下定位效果的对比和③ 版本匹配下设置下模型消耗时间的对比. 根据实验设置 (第 3.2 节) 中的描述, 版本失配对照实验中传统 IRBL 模型和 CodeBERT 模型分别采用第 3.2.1 节 (1) 和第 3.2.2 节 (3) 的设置. 版本匹配对照实验则采用第 3.2.1 节 (2) 设置下的传统 IRBL 模型与在第 3.2.2 节 (1) 和第 3.2.2 节 (2) 设置下的 CodeBERT 模型进行对比.

通过上述对比实验, 可以全面地展现 CodeBERT 模型在进行缺陷定位的效果, 它所存在的局限性以及需要改进的方向. 这些不仅有助于 CodeBERT 模型进一步优化, 也能够给类似的基于深度学习的缺陷定位模型提供新的思路.

(2) RQ2: 版本失配与数据泄漏是否会影响 CodeBERT 模型和传统 IRBL 模型的结果?

该研究问题的目的是调查在缺陷定位任务中, 版本失配与数据泄露两个问题分别对使用传统特征的 IRBL 模型和 CodeBERT 模型结果产生的影响. 该研究问题中分别对使用传统特征的 IRBL 模型和 CodeBERT 模型进行实验.

• 使用传统特征的 IRBL 模型. 由于这类模型属于无监督学习模型, 不存在数据泄露的情况, 因此只要考虑版本失配对于模型结果的影响. 该部分实验分为版本失配和版本匹配 (见第 3.1.1 节), 版本失配不需要考虑缺陷报告与定位的源代码版本是否一致, 所有的缺陷报告在同一版本的源代码上进行定位; 版本匹配则是分版本进行缺陷定位实验. 后者比前者更贴近真实的定位情况.

• 对于 CodeBERT 模型. 对该模型要考虑到数据泄露和版本失配所引起的双重影响. 设置 3 种实验设置进行对比 (第 3.2.2 节).

该部分的实验结果可以说明越贴近真实情况时模型性能可能会发生的变化. 这也反映了在以往模型评估中实验设置一些细节上的缺陷和不足, 有助于在今后验证实验时的进行完善.

(3) RQ3: 不同参数设置对 CodeBERT 模型有怎样影响?

该研究问题的目的是寻找对于 CodeBERT 模型在缺陷定位方面较优的参数设置. 对于基于深度学习的方法,

参数的设置会对实验结果造成很大的影响. 在本文中, CodeBERT 模型中的大部分参数采用原论文中[23]的缺省值, 而对于部分必须确定的参数使用在小规模数据集上进行调整达到较优结果, 再在全部的数据集上进行推广和测试. 参数主要针对两方面: ① 输入句子对涉及的文本分割值; ② 训练模型时正样本和负样本的输入比例.

文本分割值指的是输入句子对中缺陷报告和源代码文件所占的字节数. CodeBERT 的输入长度仅是固定的 512 个字节, 如何将字节数目合理地分配给缺陷报告和源代码文件, 从而可以达到较优的效果是一个值得推敲的问题. 正样本和负样本的输入比例指的是训练模型所用到的正负样本数目之间的一个比例关系. 众所周知, 缺陷一般是由项目中极少部分源代码引起的, 因此缺陷报告只与少部分源代码文件相关, 与绝大部分源代码文件不相关. 这样的现象造成了训练数据正样本和负样本的极度不平衡, 会影响模型的训练和拟合. 虽然 CodeBERT 模型中本身包含对于不平衡数据的处理, 但是用于训练的正负样本的数量比例需要针对实际项目情况进行调整, 才能有较好的效果.

该部分的实验结果通过对比不同参数设置, 可探索 CodeBERT 模型在缺陷定位任务上的最佳性能参数范围. 此外, 这些实验参数也能够为后续研究者应用 CodeBERT 模型进行缺陷定位提供帮助.

4 实验结果

本节依次详述第 3.4 节中提出的每个研究问题对应的实验结果并进行总结.

4.1 RQ1 的实验结果

表 3 展示版本失配设置下各模型缺陷定位性能的对比结果. 在该表格中, 列表头为 8 个测试项目名称, 行表头为 7 种定位模型 (6 个传统 IRBL 模型和 CodeBERT 模型). 表中加粗的数值为某个项目中各个模型能够达到的最大值.

表 3 版本失配下的排序性能对比: *MRR* 和 *MAP*

评估指标	定位模型	HBase	HIVE	LANG	MATH	WFCORE	BATCH	ROO	SEC	平均值
MRR	BugLocator	0.404	0.344	0.599	0.364	0.428	0.484	0.429	0.426	0.435
	BRTracer	0.489	**0.413**	0.618	0.373	**0.459**	0.504	0.455	0.440	0.469
	BLUiR	0.401	0.340	0.560	0.353	0.341	0.396	0.386	0.380	0.395
	AmaLgam	0.360	0.315	0.383	0.271	0.271	0.348	0.346	0.334	0.329
	BLIA	0.378	0.363	**0.824**	**0.768**	0.383	**0.578**	**0.507**	**0.681**	**0.560**
	Locus	**0.559**	**0.413**	0.619	0.377	0.446	0.475	0.412	0.562	0.483
	CodeBERT	0.002	0.001	0.706	0.032	0.009	0.032	0.134	0.040	0.120
MAP	BugLocator	0.302	0.255	0.506	0.281	0.314	0.342	0.349	0.336	0.336
	BRTracer	0.359	0.301	0.520	0.284	0.334	0.355	0.368	0.351	0.359
	BLUiR	0.298	0.248	0.522	0.296	0.255	0.299	0.313	0.325	0.320
	AmaLgam	0.267	0.231	0.357	0.228	0.203	0.263	0.281	0.286	0.265
	BLIA	0.290	0.280	**0.700**	**0.608**	0.293	0.437	**0.440**	**0.557**	**0.451**
	Locus	**0.443**	**0.308**	0.585	0.319	**0.351**	**0.445**	0.347	0.538	0.417
	CodeBERT	0.002	0.001	0.600	0.023	0.005	0.015	0.077	0.031	0.094

首先对使用传统特征的 IRBL 模型的定位效果进行分析. 可以看出, 在版本失配实验设置下, 两种较新提出的 IRBL 定位模型: BLIA (2017 年) 和 Locus (2016 年) 的缺陷定位效果较优. 平均来说, BLIA 模型的 *MRR* 值为 0.560, *MAP* 值为 0.451, 分别在 5 (和 4) 个项目上达到了最好 *MRR* (和 *MAP*) 值; 而 Locus 模型的 *MRR* 值为 0.483, *MAP* 值为 0.417, 分别在 2 (和 4) 个项目上达到了最好的 *MRR* (和 *MAP*) 值. 这与之前研究[3,6]的结论基本一致. 与此同时, AmaLgam 模型在测试项目上的定位效果最差. 平均来说, 它仅达到了 0.329 的 *MRR* 值和 0.265 的 *MAP* 值.

相比于传统的 IRBL 模型, CodeBERT 模型的缺陷定位效果很差. 传统 IRBL 定位模型的 *MRR* 平均值分布在 0.329 到 0.560 之间, *MAP* 平均值分布在 0.265 到 0.451 之间, 而 CodeBERT 模型除了在 LANG 项目上有显著定位效果 (*MAP* 和 *MRR* 数值为 0.706 和 0.600) 以外, 在其他项目上的定位效果十分低下 (*MAP* 和 *MRR* 数值大多在

0.1 以下). LANG 项目与其他项目相比, 项目中的源代码文件数量和版本数目都较少. LANG 项目的源文件数目为 305, HIVE 项目的源文件数目是 LANG 项目的 15 倍. 这些项目规模上的差距在传统的 IRBL 方法中并没有产生什么特别的影响, 但对于 CodeBERT 模型的定位效果有很大影响.

图 7 报告了各个模型在版本匹配情况下的 *MRR* 和 *MAP* 指标数值分布情况. 从左至右前 6 个箱线图为传统的 IRBL 方法 (包括 BugLocator、BRTracer、BLUiR、AmaLgam、BLIA 和 Locus), 后 2 个箱线图为 CodeBERT 方法, CodeBERT(1) 和 CodeBERT(2) 分别表示在第 3.2.2 节中第 1 种和第 2 种实验设置下 CodeBERT 模型的实验效果.

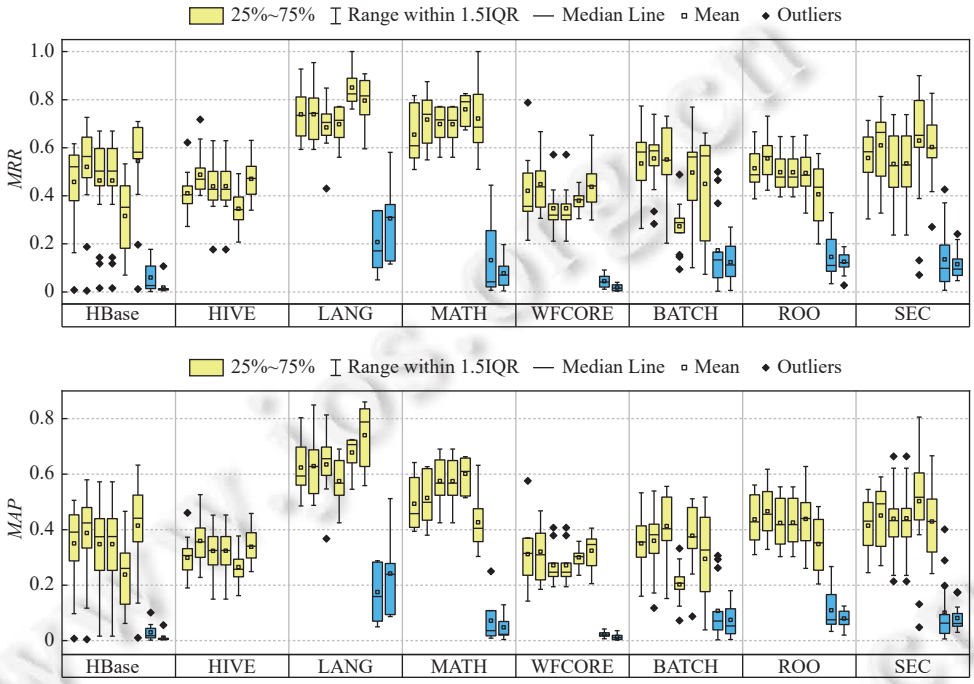


图 7 版本匹配下的排序性能对比: *MRR* 和 *MAP*

观察传统 IRBL 模型和 CodeBERT 模型的性能分布可以看出, 前 6 个箱线图明显高于后 2 个箱线图, 这说明在版本匹配的设置下, 传统的 IRBL 方法定位效果依旧明显优于 CodeBERT 模型. 在 111 个测试项目版本 (除 LANG 项目所有版本) 中, CodeBERT 模型 *MRR* 平均数值在 0.2 以下, *MAP* 平均数值在 0.1 以下. 然而, 传统 IRBL 模型的 *MRR* 平均数值大多在 0.4 以上, *MAP* 平均数值大多在 0.3 以上. 与版本失配设置类似, CodeBERT 模型唯一定位效果较好的是项目 LANG, CodeBERT(1) 和 CodeBERT(2) 的平均 *MAP* 值分别为 0.175 和 0.242; 平均 *MRR* 值分别为 0.207 和 0.305. 但同时, 各传统 IRBL 模型在该项目上也有不错的效果. 其中效果最差的 AmaLgam 方法 *MAP* 和 *MRR* 值为 0.698 和 0.575, 远超 CodeBERT 模型.

此外, 本文以项目 BATCH 为例, 比较 IRBL 方法在版本匹配下所消耗的时间和 CodeBERT 模型在第 3.2.2 节中第 3 种实验设置下所消耗的时间. 在 8 个测试项目中, BATCH 项目最大源文件数目为 1 732, 拥有 14 个版本, 属于规模大小中等的项目, 因此选取该项目对比较为合适. 表 4 报告了各个模型用于缺陷定位所消耗的时间, 消耗时间单位为 s, 加粗数值为每个版本所需的最短消耗时间. 根据统计的各个模型在 BATCH 项目上进行缺陷定位所消耗时间, AmaLgam 模型在传统 IRBL 中的平均消耗时间最短 (即 2.4 s), BLIA 模型的平均消耗时间最长 (即 21.8 s). 相比于传统 IRBL 模型, CodeBERT 模型的平均消耗时间为 5 475 s. 这个时间长度远超各种传统 IRBL 模型. 值得注意的是, CodeBERT 模型的最小消耗时间为 3 448 s, 是 IRBL 方法最大消耗时间 40.8 s 的 84.5 倍.

以上结果表明: 在大多数的测试项目中 CodeBERT 的效果完全比不上传统的 IRBL 模型. CodeBERT 模型需

要大量的计算资源和时间, 并且在源代码文件数量较大 (文件数量超过 1 000) 的项目上几乎没有定位效果, 只有对于源代码文件数目较少的测试项目集上 (如 LANG), CodeBERT 才会具有一定的定位效果. CodeBERT 的定位效果在不同规模的测试项目上差异较大, 而传统的 IRBL 模型在不同规模的测试项目上表现较为稳定.

表 4 版本匹配下各模型在项目 BATCH 上消耗时间对比 (s)

版本	BugLocator	BRTracer	BLUiR	AmaLgam	BLIA	Locus	CodeBERT(3)
1.0.0-m3	5.4	6.8	13.8	**4.5**	21.3	6.8	—
1.0.0.m4	5.0	2.8	30.4	**3.4**	22.5	4.5	7 880
1.0.0.m5	3.2	5.3	9.5	**1.7**	18.6	6.5	6 349
1.0.0.rc1	3.9	5.1	11.8	**3.2**	20.3	9.1	6 283
1.0.0.FINAL	5.3	5.8	12.7	**0.6**	18.1	6.5	6 789
1.1.0.RELEASE	3.6	4.2	11.2	**2.2**	17.9	6.0	6 493
2.0.0.M3	7.6	7.9	18.7	**3.7**	24.2	13.4	4 200
1.0.1.RELEASE	2.8	4.5	11.7	**0.5**	20.0	8.7	4 096
2.0.0.RC1	12.9	11.8	22.8	**2.0**	27.1	11.9	6 603
2.0.0.RELEASE	6.0	8.6	27.5	**5.4**	22.3	16.4	6 015
2.1.0.RELEASE	7.2	8.6	17.6	**3.5**	21.4	15.8	4 830
2.1.1.RELEASE	4.5	6.3	17.6	**1.2**	26.8	9.5	4 220
2.1.7.RELEASE	4.8	6.5	21.3	**1.4**	21.5	11.3	3 969
2.1.8.RELEASE	4.6	6.0	40.8	**1.0**	23.6	11.1	3 448
平均值	5.5	6.4	19.1	**2.4**	21.8	9.8	5 475

4.2 RQ2 的实验结果

表 5 对比了版本匹配设置下传统 IRBL 定位模型的排序性能, 并且标注了每组结果相对于版本失配设置下的变化趋势. 每个项目的具体数值为当前项目中所有版本评估指标数值的平均值, 后面的箭头表示相较于版本失配设置, 版本匹配设置下的实验结果的提高 (上箭头) 或者降低 (下箭头) 趋势. 表中加粗标注的数值为每个测试项目中所对应的最高指标数值. 可以看出, 大多数传统 IRBL 模型包括 BugLocator、BRTracer、BLUiR、AmaLgam 模型的定位性能在版本匹配设置下均有提高. 这些模型分别在 *MRR* 的指标上提高 21.1%、21.3%、30.9%、47.2%; 在 *MAP* 的指标上提高了 19.9%、19.3%、31.4% 和 46.0%. 虽然 BLIA 模型在版本匹配设置下的 *MAP* 和 *MRR* 指标有所下降, 幅度约为 4.6% 和 5.9%, 但是这个差距并不是很大, 可能与所选择的测试项目有关. 而 Locus 模型在 *MRR* 上提高了 12.5%, 却在 *MAP* 上降低了 2.0%. 这说明版本匹配设置下, Locus 模型更容易将首个正确的推荐结果排到列表的前面.

表 5 版本失配设置与版本匹配设置下传统 IRBL 模型的排序性能对比: *MRR* 和 *MAP*

评价指标	定位模型	HBase	HIVE	LANG	MATH	WFCORE	BATCH	ROO	SEC	平均值
MRR	BugLocator	0.457↑	0.410↑	0.739↑	0.581↑	0.420↓	0.534↑	0.514↑	0.557↑	0.527↑
	BRTracer	0.520↑	**0.488**↑	0.739↑	0.637↑	**0.448**↓	**0.554**↑	**0.555**↑	0.610↑	**0.569**↑
	BLUiR	0.463↑	0.439↑	0.684↑	0.620↑	0.348↑	0.551↑	0.498↑	0.532↑	0.517↑
	AmaLgam	0.463↑	0.439↑	0.698↑	0.620↑	0.348↑	0.273↓	0.498↑	0.534↑	0.484↑
	BLIA	0.315↓	0.346↓	**0.850**↑	**0.760**↓	0.380↓	0.496↓	0.494↓	**0.630**↓	0.534↓
	Locus	**0.545**↓	0.471↑	0.796↑	0.641↑	0.437↓	0.449↓	0.406↓	0.602↑	0.543↑
MAP	BugLocator	0.350↑	0.298↑	0.624↑	0.438↑	0.312↓	0.350↑	0.437↑	0.414↑	0.403↑
	BRTracer	0.388↑	**0.359**↑	0.629↑	0.456↑	0.320↓	0.359↑	**0.466**↑	0.450↑	**0.429**↑
	BLUiR	0.347↑	0.323↑	0.635↑	0.511↑	0.272↑	**0.413**↑	0.424↑	0.439↑	0.421↑
	AmaLgam	0.347↑	0.324↑	0.575↑	0.510↑	0.272↑	0.202↓	0.425↑	0.441↑	0.387↑
	BLIA	0.238↓	0.265↓	0.678↓	**0.601**↓	0.300↑	0.378↓	0.438↓	**0.502**↓	0.425↓
	Locus	**0.414**↓	0.338↑	**0.740**↑	0.379↑	**0.323**↓	0.294↓	0.348↑	0.429↓	0.408↓

文献 [19] 中也对比了 6 种传统 IRBL 模型的效果, 在版本匹配的情况下 6 种模型的 *MAP* 指标提高了 19.9% 到 28.2%, *MAP* 指标提高了 17.8% 到 24.5%. 虽然在数据集进行筛选和过滤之后, 本文实验结果数据与文献 [19] 中有细微偏差 (除 Locus 外, 其他模型的 *MAP* 和 *MRR* 有 3%–16% 的增幅), 但是总体上有相似的结论: 在版本匹配的情况下, IRBL 模型的定位性能都有所提高. 然而, 与文献 [19] 不同的是, 我们发现: 在 6 个使用传统特征的模型中, BRTracer 表现最好, 而不是 Locus. 这部分结果的差异主要由于本文对于文献 [19] 中的数据集进行了过滤和筛选, 可能去除了 Locus 表现较优的测试项目. 由表 5 可以看出, 虽然 Locus 在 *MRR* 指标上表现比不过 BRTracer, 但是在 *MAP* 指标上是 6 个模型中最优的, 因此 Locus 也是具有不错的定位性能, 这也符合文献 [19] 中的结论.

表 6 报告了版本匹配设置下 CodeBERT 定位模型的排序性能, 并且标注了每组结果相对于版本失配设置下的变化趋势. 该表格的结构与表 5 类似. 其中, CodeBERT(1) 和 CodeBERT(2) 中的数值分别代表 CodeBERT 模型在第 1 种和第 3 种实验设置上的定位效果. 可以看出, 对于定位效果较好的 LANG 项目, 第 3 种实验设置下的结果远超第 1、第 2 种设置下的实验结果, *MRR* 指标下降了 70.7% 和 56.8%, *MAP* 指标下降了 70.8% 和 59.7%, 这说明传统的测试方法可能存在由于数据泄露而导致定位结果的乐观, 使得对于深度学习定位模型的评估不准确. 模型在训练阶段获取了“未来”的信息, 使得对模型评估有乐观性的误导. 该结果也证明了数据泄漏因素深刻影响模型性能. 对于 HIVE 项目, 划分测试方法的评估指标均在 0.001 左右, 说明 CodeBERT 在该项目上几乎不存在定位效果. 对于其他测试项目, CodeBERT 模型有一定的定位效果 (*MAP* 和 *MRR* 的数值超过 0.1), 在这种情况下, 版本匹配设置下的结果比版本失配设置的结果要好, 这与传统的定位模型所得出的结论一致.

表 6 版本失配与版本匹配下 CodeBERT 的排序性能对比: *MRR* 和 *MAP*

评价指标	定位模型	HBase	HIVE	LANG	MATH	WFCORE	BATCH	ROO	SEC	平均值
MRR	CodeBERT(1)	**0.060**↑	—	0.207↓	**0.132**↑	**0.044**↑	**0.173**↑	**0.145**↑	**0.135**↑	**0.128**↑
	CodeBERT(2)	0.019↑	—	**0.305**↓	0.077↑	0.018↑	0.123↑	0.125↑	0.115↑	0.111↓
MAP	CodeBERT(1)	**0.030**↑	—	0.175↓	**0.072**↑	**0.022**↑	**0.107**↑	**0.110**↑	**0.101**↑	**0.088**↓
	CodeBERT(2)	0.010↑	—	**0.242**↓	0.048↑	0.013↑	0.074↑	0.079↑	0.081↑	0.078↓

此外, Zhang 等人[38]发现深度神经网络很容易拟合随机标签. 对于训练误差, 模型最后的有效性与训练时采用的标签对错没有关系. 这个因素也会导致深度学习模型对于错误实验设置的不敏感. 因此, 对于 CodeBERT 模型可以在与真实情况完全偏差的实验设置中, 在 LANG 项目上获得较好的实验结果, 可能与其深度学习模型的特性有关.

根据实验对比得出以下结论.

(1) 对于传统的 IRBL 模型, 虽然版本与版本之间的定位效果相差较大, 但整体而言, 版本匹配的实验设置模型下的结果比版本失配的实验设置下的模型结果要好. 版本失配会拉低定位模型的整体效果. 版本匹配的实验设置比版本失配的实验设置更贴近真实的情况. 因此, 传统的缺陷定位模型 (如 BugLocator, BRTracer 等) 的真实定位性能其实是被低估了.

(2) 对于 CodeBERT 模型, 可以发现在对于部分评估指标较低的测试项目, 第 1 种实验设置 (邻近版本的测试) 下实验结果是 3 种实验设置下效果最优的. 对于相对评估指标较高的测试项目, 第 3 种实验设置 (传统的测试) 下的效果最好.

4.3 RQ3 的实验结果

图 8 为多种参数组合在 LANG 项目上的性能, 横坐标为参数的组合, 方形点曲线为 *MRR* 指标数值, 圆形点曲线为 *MAP* 指标数值. 参数组合中第 1 个数值为文本分割后缺陷报告所占的字节数, 第 2 个数值为输入负样本对正样本的倍数. 由于输入的句子对由缺陷报告和源代码组成. 同一个缺陷报告, 句子对与句子对的差距体现在后接的源代码内容, 因此在组成句子对是尽可能保证源代码的部分保留较长, 更好捕捉输入句子对的特征差异. 因此, 输入缺陷报告长度选择 25 个字节和 50 个字节作为比较. 由于 CodeBERT 模型中考虑到了正负样本不均衡的处理, 但不清楚该处理效果在缺陷定位问题上的具体影响. 因此, 训练数据将所有正样本和负样本顺序打乱. 由于训练正

样本数量较少, 因此对正样本不做删减, 根据正样本的数量按一定比例随机抽取负样本进行训练. 设置 1 倍、2 倍、3 倍的负样本数量进行训练, 得出在哪种情况下模型训练效果较理想. 训练设置参照 CodeBERT 模型的第 3 种实验设置, 对于每种参数搭配都以划分法得出 *MAP*/*MRR* 数值, 以 *MAP*/*MRR* 的数值高低作为参数搭配的参考.

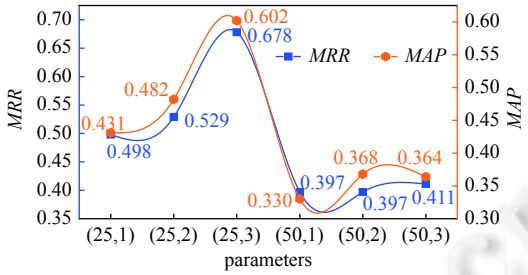


图 8 多种参数组合在 LANG 项目上的性能

根据实验结果, 在参数组合为 (50, 1) 下, 模型在项目 LANG 上定位效果最差, *MRR* 的数值为 0.397, *MAP* 的数值为 0.330; 而在参数组合为 (25, 3) 的情况下, 效果最好, *MRR* 数值为 0.678, *MAP* 数值为 0.602. 因此, 本文中所有 CodeBERT 模型实验均采用 (25, 3) 组合参数.

实验结果表明: 对于每一组缺陷报告和源代码文件的输入对, 512 个字节中 25 个字节分配给缺陷报告文本, 剩下的字节分配给源代码文本. 在训练模型时输入的负文本数量和正文本数量比例为 3:1 时, 模型的拟合和测试效果是最优的.

5 讨 论

本节主要讨论先前的相关研究工作[19,20]的不足之处以及本文中做出的相应改进工作.

5.1 与文献 [19] 工作的比较

Lee 等人[19]在一个大型数据集 (包括 61 431 个 Java 文件和 9 459 个缺陷报告) 上通过设置对照实验分析了版本失配因素对传统的 IRBL 模型的影响. 该研究中仍存在以下两点不足之处.

(1) 实验项目选取不当. 文献 [19] 中的用到的实验数据集并不完全合适, 其中存在某些项目包含缺陷报告过少的情况. 如 WFMP 项目只包含 3 个缺陷报告而 WFARQ 项目中仅包含 1 个缺陷报告. 当项目中的缺陷报告数目过少时, 性能值 *MAP* 和 *MRR* 的计算会受单个缺陷报告的影响大, 有可能会导致它们的评估值过低或者过高. 例如, 文献 [19] 中报告的很多定位模型 (如 BLUiR 和 AmaLgam) 在 WFARQ 项目上的 *MRR* 和 *MAP* 值均为 1. 这明显不能代表该模型的真实性能. 因此, 为了消减不恰当的实验项目对分析模型优劣的结论产生影响, 本文设计了详细的数据筛选与划分步骤来过滤不合适的测试项目, 重新在适当的 8 个项目上验证版本匹配因素对各类 IRBL 模型的影响.

(2) 研究对象类型单一. 文献 [19] 中调研的对象只包括了使用传统特征的定位模型, 这类模型虽然在构造上有不同之处, 但本质上属于同一类模型. 在近年来, 基于深度学习的缺陷定位模型广受关注[13−15]. 因此, 本文中增加了对于基于深度学习特征的缺陷定位模型的分析, 设计了 CodeBERT 模型. 主要分析版本失配和数据泄露两个因素对该模型的影响, 并且将 CodeBERT 模型与传统模型进行对比, 为后续研究者提供了一个初步的探索.

5.2 与文献 [20] 工作的比较

Tu 等人[20]在其工作中研究的“数据泄露”的概念, 指的是定位模型使用了字段被更改的缺陷报告 (泄露的数据) 而不是初始提交时的缺陷报告进行缺陷定位, 使模型的评估不客观. 在该论文中, 假定缺陷定位的实际应用场景是在缺陷报告初始提交时发生的, 此时使用在后续时间中字段发生更改后的缺陷报告进行定位, 会导致使用了“未来”的信息, 产生了数据泄露. 作者在 Eclipse 数据集上验证数据泄露对于缺陷定位的影响. 相比于该研究, 本文的工作有如下两方面的不同.

(1) 调研由时间顺序紊乱引发的数据泄露. 在实践中, 在一个用户提交缺陷报告给缺陷追踪数据库后, 开源项目管理人员审核缺陷报告, 之后分配给开发者解决. 在开发者拿到缺陷报告后, 此时才进行缺陷定位. 由上述描述可见, 缺陷定位是针对开发者而不是常规意义下的用户的. 在开发者拿到缺陷报告之前, 用户或者开源项目管理人员是可以对缺陷报告的相关字段进行修改的. 因此, 在我们研究中, “数据泄露”指在缺陷定位实际发生的时间点使用了该时间点之后的数据来训练神经信息检索模型, 即在训练时间点使用了还没有出现的缺陷报告进行模型的训练.

(2) 使用更多的数据集进行分析. 文献 [20] 中只在一个数据集 Eclipse 上验证了所定义的数据泄露的影响, 不清楚其结论在多大程度上能推广其他项目上. 因此, 根据本文中所定义数据泄露, 我们设置对比实验, 使用了 8 个开源项目对神经信息检索模型进行验证, 提高结论的泛化能力.

6 实验效度威胁

本节主要对实验过程中可能影响实验结论有效性的威胁因素进行分析, 包括构建效度、内部效度和外部效度 3 个方面.

6.1 构建效度

构建效度是指实验中涉及的独立变量和依赖变量的获取过程的有效性. 本文中独立变量是指缺陷报告和代码文件, 依赖变量是指缺陷报告与对应缺陷代码之间的链接关系, 也就是实验中用到的数据集. 该数据集收集于 Apache2、Spring3 和 JBoss4 的软件社区, 这些项目是由 Lee 等人[19]进行收集并且验证过. 表 2 中的每个测试项目均是从相应的问题跟踪系统 (issue tracking systems) 中收集缺陷报告. 在所有可用的缺陷报告中, 只保留被开发人员明确归类为 bug 的缺陷报告, 并且是标记为已修复的缺陷报告. 同时, 实验中用的缺陷报告是通过提取出最初提交时的可用信息, 例如, 摘要 (或标题)、描述等, 忽略了注释等其他信息.

为了确保使用符合真实场景的数据集, 我们对原始数据集进行了筛选和划分. 在版本匹配的实验设置中, 根据缺陷报告所获取的版本信息, 通过 git 将源代码回滚到相应的版本. 有些缺陷报告的版本无法找到相对应的源代码版本. 因此, 为了尽可能确保每个版本都能获得相对应的源代码版本, 人工对每一个缺陷报告的版本进行检查, 过滤其中一些实在无法找到对应源代码文件的版本. 由于实验设置对于实验数据的限制, 可能会导致最后实验结果的差异.

在 CodeBERT 模型的实验设置中, 还要通过版本和时间划分训练集和测试集. 这里所选的时间是通过 git 获取 tag 标签的提交时间作为基准, 这导致了可能测试集和训练集划分得不合理性 (例如训练集数据过少等情况). 虽然本文尽力保证实验设置得客观性, 然而受到数据集的限制, 不能保证每一种模型都完全达到完全理想的定位效果.

关于版本匹配问题, 本文中涉及两方面因素无法避免. 一方面, 在用户提交缺陷报告过程中, 可能会存在缺陷报告版本信息缺失的情况. 某些缺陷报告的版本信息是后期开发人员进行完善的[20]. 因此这些缺陷报告无法得知真正出问题的代码版本. 另一方面, 项目版本的更迭与缺陷报告的提交是并行发生的, 所以很难非常精确地将缺陷报告定位到最匹配的代码中. 本文中采用的是使用 git 所记录的版本信息和最新提交时间作为划分点, 虽然一定程度上可以改善版本失配的情况, 但可能离完全理想的版本匹配的情况有距离, 无法完全解决该问题. 但是, 在大多数情况下, 同一版本的代码差距不会过于悬殊, 因此在版本匹配的情况下进行缺陷定位, 可以较好降低版本失配对定位结果所带来的影响.

6.2 内部效度

内部效度是指在实验中存在会对结果准确性产生影响的因素. 本实验中的影响因素主要指各个模型的实现准确性. 本文对 6 种使用传统特征 IRBL 模型和一个基于 CodeBERT 预训练模型的定位模型进行实证研究. 因为使用传统特征的 IRBL 模型的代码已经被其原作者开源, 所以这 6 种模型的实现应该是可靠的. 本文中它们对应的实验结果与其他研究者的结果数据虽略有差异, 但总体结论基本一致, 其结果能够反映每个模型的真实性能表现.

对于使用深度学习的 CodeBERT 模型来说, 原作者提供了模型的 API 接口[23], 本文在其基础上实现定位模型. 它的性能有部分依赖于模型的参数和测试数据集. 在本文的实验中, 主要目标是比较在不同实验设置下不同模

型的定位性能, 而不是寻找 CodeBERT 模型的最优状态. 因此, 在对于 CodeBERT 模型的测试中, 本文客观地保证参数的一致性, 尽可能让不同的实验模型在同一量纲上进行对比. 此外, 与 CodeBERT 模型的实验结果有一定的随机性, 即在保证参数一致的情况下, 每次训练出来的结果也不一定相同. 因此, 在本文的实验中, 由于涉及 8 个项目、120 个版本、3 种实验设置, 训练与测试的工作量较大, 因此只能使用有限的资源保证模型收敛的情况下尽可能保证实验结果的客观性. 但实验之外, 也不排除 CodeBERT 模型拥有更高缺陷定位性能的可能性.

6.3 外部效度

外部效度是指本文的实验结论是否可以推广到其他的定位模型和项目. 本文中用到的实验项目完全来自开放软件社区中的 8 个开源的 Java 项目. 这些项目的开发过程都比较规范, 而且对于开发过程以及其过程中的缺陷都有较为详细的信息记录. 因此, 本文的结论有一定的代表性. 然而, 对于 IRBL 的方法来说, 每种定位模型都有各自的偏好和优势, 所以在本文中得出的结论也许不一定完全适用于其他所有的项目. 比如 CodeBERT 模型对于不同的项目, 定位效果差异比较大. 因此对于 CodeBERT 模型是否可以应用于缺陷定位任务, 还需要更多的实验数据来进行验证.

CodeBERT 模型并不是完全为解决缺陷定位问题而设计的, 在其性能上比不上其他基于深度学习的缺陷定位模型, 如 2016 年提出的 NPCNN[39]、2017 年提出的 LS-CNN[40]和 2020 年提出的 GC-CNN[41], 而这些方法均未开源而无法完全复现, 所以 CodeBERT 上的研究结论是否能推广到所有的基于深度学习的缺陷定位模型需要进一步的研究论证.

7 总结与未来工作

如何通过缺陷报告来定位项目中的缺陷一直是一个热门问题. 为了能够使用更自动化的方法帮助软件开发者能快速定位缺陷, 研究者提出了许多模型. 然而对于各种 IRBL 模型的评估却在实验设置上存在问题, 影响最后模型评估的客观性. 本文通过不同的实验设置, 在模拟真实的环境下, 评估 6 种传统的 IRBL 缺陷定位模型 (BugLocator、BRTracer、BLUiR、AmaLgam、BLIA、Locus) 和深度学习模型 CodeBERT 在 8 个测试项目上的缺陷定位效果. 根据实验结果, 得出以下结论.

(1) 对传统的非深度学习的定位模型而言, 其真实的定位性能被低估.

(2) CodeBERT 模型之间用于缺陷定位效果受到测试数据集的限制, 版本数目越多, 源代码数目越大的项目, 缺陷定位效果越差.

(3) CodeBERT 模型的缺陷定位效果会受到版本失配和数据泄露两方面的双重影响. 在定位情况较好的项目中, 数据泄露会导致定位效果过于乐观. 同时, 在定位情况不理想的项目中, 版本匹配的设置可以更好捕捉缺陷报告和源代码之间的联系, 使定位效果优于简单的划分验证的设置.

在未来研究中, 我们将会进行以下两个方面的工作.

(1) 本文的实验结果验证了在实验中版本失配和数据泄露问题会对基于信息检索的缺陷定位模型的评估产生影响. 然而, 这两个因素是否会对其他类型的缺陷定位模型产生影响需要进一步的验证. 因此, 在未来工作中我们想要分析其他定位模型是否会因为版本失配和数据泄露而导致定位结果不准确.

(2) 本文是对于 CodeBERT 模型在缺陷定位问题上的初步尝试, 因此存在较大的提升空间. 首先, 在特征提取方面, 输入长度的限制导致特征提取的不完善, 以此影响模型的后续性能. 在后续工作中, 我们考虑在 CodeBERT 模型前加入一个新的神经网络 (如 LSTM) 用于提取特征. 另外, CodeBERT 模型对于源代码数目较多的大型项目效果较差, 且消耗时间巨大. 如何用更有效率的方法提高大规模项目的训练结果也是我们的一个未来研究目标.

致谢 感谢期刊编辑以及匿名审稿专家提出的宝贵意见.

References:

[1] Pressman RS. Software Engineering: A Practitioner’s Approach. 7th ed., New York: McGraw-Hill, 2010. 437−443.

[2] Wong CP, Xiong YF, Zhang HY, Hao D, Zhang L, Mei H. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and Evolution. Victoria: IEEE: 2014. 181−190. [doi: 10.1109/ICSME.2014.40]

[3] Wen M, Wu RX, Cheung SC. Locus: Locating bugs from software changes. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 262−273. [doi: 10.1145/2970276.2970359]

[4] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering. Silicon Valley: IEEE, 2013. 345−355. [doi: 10.1109/ASE.2013.6693093]

[5] Wang SW, Lo D. Version history, similar report, and structure: Putting them together for improved bug localization. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 53−63. [doi: 10.1145/2597008.2597148]

[6] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 2017, 82: 177–192. [doi: 10.1016/j.infsof.2016.11.002]

[7] Guo ZQ, Zhou HC, Liu SR, Li YH, Chen L, Zhou YM, Xu BW. Information retrieval based bug localization: Research problem, progress, and challenges. Ruan Jian Xue Bao/Journal of Software, 2020, 31(9): 2826–2854 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[8] Zhao F. Localizing relevant source code files for bug reports—An effort-aware effectiveness evaluation [MS. Thesis]. Nanjing: Nanjing University, 2016 (in Chinese with English abstract).

[9] Zhang Y, Liu JK, Xia X, Wu MH, Yan H. Research progress on software bug localization technology based on information retrieval. Ruan Jian Xue Bao/Journal of Software, 2020, 31(8): 2432−2452 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6081.htm [doi: 10.13328/j.cnki.jos.006081]

[10] Li ZL, Chen X, Jiang ZW, Gu Q. Survey on information retrieval-based software bug localization methods. Ruan Jian Xue Bao/Journal of Software, 2021, 32(2): 247–276 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6130.htm [doi: 10.13328/j.cnki.jos.006130]

[11] Panichella A, Dit B, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A. Parameterizing and assembling IR-based solutions for SE tasks using genetic algorithms. In: Proc. of the 23rd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering. Osaka: IEEE, 2016. 314–325. [doi: 10.1109/SANER.2016.97]

[12] Chaparro O, Florez JM, Marcus A. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empirical Software Engineering, 2019, 24(5): 2947–3007. [doi: 10.1007/s10664-018-9672-z]

[13] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Program Comprehension. Buenos Aires: IEEE, 2017. 218−299. [doi: 10.1109/ICPC.2017.24]

[14] Xiao Y, Keung J. Improving bug localization with character-level convolutional neural network and recurrent neural network. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 703–704. [doi: 10.1109/APSEC.2018.00097]

[15] Xiao Y, Keung J, Bennin KE, Mi Q. Machine translation-based bug localization technique for bridging lexical gap. Information and Software Technology, 2018, 99: 58–61. [doi: 10.1016/j.infsof.2018.03.003]

[16] Xiao Y, Keung J, Bennin KE, Mi Q. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology, 2019, 105: 17–29. [doi: 10.1016/j.infsof.2018.08.002]

[17] Polisetty S, Miranskyy A, Başar A. On usefulness of the deep-learning-based bug localization models to practitioners. In: Proc. of the 15th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. Recife: ACM, 2019. 16−25. [doi: 10.1145/3345629.3345632]

[18] Zhou J, Zhang HY, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the 34th Int'l Conf. on Software Engineering. Zurich: IEEE, 2012. 14−24. [doi: 10.1109/ICSE.2012.6227210]

[19] Lee J, Kim D, Bissyandé TF, Jung W, Le Traon Y. Bench4BL: Reproducibility study on the performance of IR-based bug localization. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Amsterdam: ACM, 2018. 61−72. [doi: 10.1145/3213846.3213856]

[20] Tu FF, Zhu JX, Zheng QM, Zhou MH. Be careful of when: An empirical study on time-related misuse of issue tracking data. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 307−318. [doi: 10.1145/3236024.3236054]

[21] Tu FF, Zhou MH. Data quality problems in software development activity data. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1522−1531 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5727.htm [doi: 10.13328/j.cnki.jos.005727]

[22] Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L. TESSERACT: Eliminating experimental bias in malware classification across space and time. In: Proc. of the 28th USENIX Conf. on Security Symp. Santa Clara: USENIX Association, 2019. 729−746. [doi:

10.5555/3361338.3361389]

[23] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. Virtual Event: Association for Computational Linguistics, 2020. 1536−1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[24] Strohman T, Metzler D, Turtle H, Croft WB. Indri: A language model-based search engine for complex queries. In: Proc. of Int'l Conf. on Intelligence Analysis. 2004. 2−6.

[25] Dit B, Moritz E, Linares-Vásquez M, Poshyvanyk D. Supporting and accelerating reproducible research in software maintenance using TraceLab component library. In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Eindhoven: IEEE, 2013. 330−339. [doi: 10.1109/ICSM.2013.44]

[26] Mitra B, Craswell N. An introduction to neural information retrieval. Foundations and Trends® in Information Retrieval, 2018, 13(1): 1–126. [doi: 10.1561/1500000061]

[27] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[28] Liu YH, Ott M, Goyal N, Du JF, Joshi M, Chen DQ, Levy O, Lewis M, Zettlemoyer L, Stoyanov V. RoBERTa: A robustly optimized BERT pretraining approach. arXiv:1907.11692, 2019.

[29] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: NIPS, 2017. 5998−6008. [doi: 10.5555/3295222.3295349]

[30] Clark K, Luong MT, Le QV, Manning CD. ELECTRA: Pre-training text encoders as discriminators rather than generators. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: OpenReview.net, 2020.

[31] Kaufman S, Rosset S, Perlich C, Stitelman O. Leakage in data mining: Formulation, detection, and avoidance. ACM Trans. on Knowledge Discovery from Data, 2012, 6(4): 15. [doi: 10.1145/2382577.2382579]

[32] Kaufman S, Rosset S, Perlich C. Leakage in data mining: Formulation, detection, and avoidance. In: Proc. of the 17th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Diego: ACM, 2011. 556−563. [doi: 10.1145/2020408.2020496]

[33] Brownlee J. Data leakage in machine learning. 2016. https://machinelearningmastery.com/data-leakage-machine-learning/

[34] Wikipedia. https://en.wikipedia.org/wiki/Leakage_(machine_learning)

[35] Koyuncu A, Bissyandé TF, Kim DS, Liu K, Klein J, Monperrus M, Le Traon Y. D&C: A divide-and-conquer approach to IR-based bug localization. arXiv:1902.02703, 2019.

[36] Akbar SA, Kak AC. A Large-scale comparative evaluation of IR-based tools for bug Localization. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 21−31. [doi: 10.1145/3379597.3387474]

[37] Sangle S, Muvva S, Chimalakonda S, Ponnalagu K, Venkoparao VG. DRAST—A deep learning and AST based approach for bug localization. arXiv:2011.03449, 2020.

[38] Zhang C, Bengio S, Hardt M, Recht B, Vinyals O. Understanding deep learning requires rethinking generalization. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017. 24−26.

[39] Huo X, Li M, Zhou ZH. Learning unified features from natural and programming languages for locating buggy source code. In: Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence. New York: IJCAI/AAAI Press, 2016. 1606−1612.

[40] Huo X, Li M. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. Melbourne: IJCAI.org, 2017. 1909−1915. [doi: 10.24963/ijcai.2017/265]

[41] Huo X, Li M, Zhou ZH. Control flow graph embedding based on multi-instance decomposition for bug localization. Proc. of the AAAI Conf. on Artificial Intelligence, 2020, 34(4): 4223–4230. [doi: 10.1609/aaai.v34i04.5844]

附中文参考文献:

[7] 郭肇强, 周慧聪, 刘释然, 李言辉, 陈林, 周毓明, 徐宝文. 基于信息检索的缺陷定位: 问题、进展与挑战. 软件学报, 2020, 31(9): 2826–2854. http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[8] 赵斐. Bug报告的相关源代码文件定位——一个工作量感知的有效性评价 [硕士学位论文]. 南京: 南京大学, 2016.

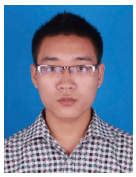
[9] 张芸, 刘佳琨, 夏鑫, 吴明晖, 颜晖. 基于信息检索的软件缺陷定位技术研究进展. 软件学报, 2020, 31(8): 2432–2452. http://www.jos.org.cn/1000-9825/6081.htm [doi: 10.13328/j.cnki.jos.006081]

[10] 李政亮, 陈翔, 蒋智威, 顾庆. 基于信息检索的软件缺陷定位方法综述. 软件学报, 2021, 32(2): 247−276. http://www.jos.org.cn/1000-9825/6130.htm [doi: 10.13328/j.cnki.jos.006130]

[21] 涂菲菲, 周明辉. 软件开发活动数据的数据质量问题. 软件学报, 2019, 30(5): 1522−1531. http://www.jos.org.cn/1000-9825/5727.htm [doi: 10.13328/j.cnki.jos.005727]

周慧聪(1996－), 女, 硕士, 主要研究领域为缺陷定位.

李言辉(1981－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为软件演化与维护, 软件测试.



郭肇强(1994－), 男, 博士生, 主要研究领域为软件度量, 缺陷定位.

陈林(1971－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为软件分析测试.

梅元清(1980－), 男, 博士生, 副教授, 主要研究领域为软件度量, 缺陷预测.

周毓明(1974－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件维护, 测试与分析.